Chapter 3
Message in a Bottle
Programming Loops in C
The Plan

- Variable declarations
- DO loops
- FOR loops
- Curious loops
- Array declaration and initialization
- Sending a message (to the world?)
  - Using the Explorer 16 board (optional)
  - Using the PIC32 Starter Kit
Variable Declarations

- **signed**
  
  ```c
  // declaration
  char  c;           // -128..127
  short s;          // -32768..32767
  int   i;          // -2147483648..2147483647
  long  l;          // -2147483648..2147483647
  ```

- **unsigned**
  
  ```c
  // declaration
  unsigned char  c;           // 0..255
  unsigned short s;          // 0..65535
  unsigned int   i;          // 0..4294967295
  unsigned long  l;          // 0..4294967295
  ```
Variable Declarations

- 64-bit integer types
  
  // declaration
  // MPLAB C32 range of values
  long long l; // ranges from $-2^{63}$ to $+2^{63}-1$
  unsigned long long l; // ranges from 0 to $+2^{64}$

- Floating-Point types
  
  float f; // 32-bit floating point
  double d; // 64-bit floating point
  long double d; // 64-bit floating point

  // difference between double and long double?
  // depends on processor architectures
  // many x86 systems use 80-bit for long double
  // PIC32 treats long double like double
DO Loops

do {
    // your code here...
} while (x);

- The exit test is performed at the end of the loop, after the body (i.e. enclosed by { }) has been executed at least once.
FOR Loops

- Assuming “i” is defined as an integer-type variable

```c
for (i=0; i<5; i=i+1)
{
    // insert your code here ...
    // it will be executed for i=0, 1, 2, 3, 4 only
}
```

- Works like a WHILE loop, except the exit test is performed **before** the loop body.

- It is equivalent to:

```c
i = 0;                      // initialize the index/counter
while (i<5)                // to be executed for i= 0, 1, 2, 3, 4
{
    // insert your code here ...
    // to be executed for i= 0, 1, 2, 3, 4
    i = i+1;                    // increment
}
More Ways to Write the Main Loop

- **Using a DO loop**
  ```c
  // main application loop
  do {
      ... // insert your main loop here...
  } while (1)
  ```

- **Using a FOR loop**
  ```c
  // main application loop
  for ( ; 1; )
  {
      ... // insert your main loop here...
  }
  ```
Increments and Decrements

- **Incrementing a variable**
  
  ```
  i++;
  ```
  is equivalent to:
  ```
  i = i + 1;
  ```

- **Decrementing a variable**
  
  ```
  i--;
  ```
  is equivalent to:
  ```
  i = i - 1;
  ```
Another Curious Loop

do{
    // your code here...
} while (0);

- The loop is executed once and only once.
Arrays

- Declarations

char c[10];    // declares c as an array of 10 x 8-bit integers
short s[10];   // declares s as an array of 10 x 16-bit integers
int i[10];     // declares i as an array of 10 x 32-bit integers

- This counts from 0.. N-1

- Some usage examples:

  a = c[0];     // copy the value of the 1st element of c into a
  c[1] = 123;   // assign the value 123 to the second element
  i[2] = 12345; // assign the value 12345 to the third element
  i[3] = 123*i[4]; // compute 123 x the value of the fifth element
Using FOR Loops with Arrays

- This is the most common use of the FOR loop: sequential access to the elements of an array

```c
int a[10];    // array of 10 int a[0], a[1], a[2]..a[9]
int i;        // integer to be used as the loop index
for (i=0; i<10; i++)
{
    a[i] = 1;
}
```
Sending a Message

```c
#include <p32xxxx.h>
// 1. define timing constants
#define SHORT_DELAY   400
#define LONG_DELAY     3200
// 2. declare and initialize an array with the message bitmap
char bitmap[30] = {
  0xff, // H
  0x08,
  0x08,
  0xff,
  0,
  0,
  0xff, // E
  0x89,
  0x89,
  0x81,
  0,
  0,
  0xff, // L
  0x80,
  0x80,
  0x80,
  0,
  0,
  0xff, // L
  0x80,
  0x80,
  0x80,
  0,
  0,
  0x7e, // O
  0x81,
  0x81,
  0x81,
  0x7e,
  0,
  0
};
```
Sending a Message

// 3. the main program
main()
{
  // disable the JTAG port
  DDPCONbits.JTAGEN = 0;
  // 3.1 variable declaration
  int i;                   // i is the index
  // 3.2 initialization
  TRISA = 0xff00;         // PORTA pins connected to LEDs are outputs
  T1CON = 0x8030;         // TMR1 on, prescale 1:256 Tpb=36MHz
  PR1 = 0xFFFF;           // max period (not used)
  // 3.3 the main loop
  while(1)
  {
    // 3.3.1 display loop, hand moving to the right
    for(i=0; i<30; i++)
    {
      // update the LEDs
      PORTA = bitmap[i];
      // short pause
      TMR1 = 0;
      while (TMR1 < SHORT_DELAY)
      {
      }
    } // for i
    // 3.3.2 long pause, hand moving back to the left
    // turn LEDs off
    PORTA = 0;
    // long pause
    TMR1 = 0;
    while (TMR1 < LONG_DELAY)
    {
    }
  } // main loop
} // main
Testing with the Logic Analyzer

1. Build the project using the Project Build check list
2. Open the Logic Analyzer window
3. Click on the Channel button to add, in order, all the I/O pins from RA0 to RA7 connected to the row of LEDs.
4. The MPLAB SIM Setup and Logic Analyzer Setup checklists will help you make sure that you don’t forget any detail.
5. Go back to the editor window and set the cursor on the first instruction of the 3.3.2 section
6. Right click to select the context menu and choose the Run to Cursor command. This will let the program execute the entire portion containing the message output (3.3.1) and will stop just before the long delay.

As soon as the simulation halts on the cursor line, you can switch to the Logic Analyzer window and verify the output waveforms.
Logic Analyzer Output

Di Jasio - Programming 32-bit Microcontrollers in C
Testing with Explorer16 (Optional)

- **Requires:**
  - Explorer16 demonstration board
  - MPLAB REAL ICE programmer and debugger

1. **Use the Setup checklist**
2. **Use the Device Configuration checklist to verify that the device configuration bits are properly set for use with the Explorer16 demonstration board**
3. **Use the Programming checklist to program the PIC32 in circuit**

- **After dimming the light a bit in the room, you should be able to see the message flashing as you “shake” the board.**
  - Consider adjusting the timing constants for optimal speed
  - Try the values 200 and 1600, respectively for the short and long delays
Lacking enough LEDs (only 3 are available), we use Morse code to send the message instead:

```c
#include <p32xxxx.h>
// 1. define timing constant
#define DOT_DELAY 18000
// 2. declare and initialize an array with the message bitmap
char bitmap[] = { // 1,0 = dot; 1,1,1,0 = bar; 0,0 = space
    // H: ....
    1,0,1,0,1,0,1,0,0,0,
    // E: .
    1,0,0,0,
    // L: .--
    1,0,1,1,0,1,0,0,0,
    // O: ---
    1,0,1,1,0,1,0,0,0,
    // end of “HELLO”
    0,0,0,0,0
};
```
Testing with PIC32 Starter Kits

// 3. the main program
main()
{
    // 3.1 variable declaration
    int i;          // i is the loop counter

    // 3.2 SFR initializations
    TRISD = 0;      // PORTD as output (to the LEDs)
    T1CON = 0x8030; // Timer1 on, prescale 1:256, PB=36MHz
    PR1 = 0xFFFF;   // max period (not used)

    // 3.3 main infinite loop
    while(1)
    {
        // 3.3.1 display one Morse letter at a time
        for(i=0; i<sizeof(bitmap); i++)
        {
            PORTD = bitmap[i]; // PORTD gets it

            // short pause to display PORTD output
            TMR1 = 0;
            while (TMR1 < DOT_DELAY) {}
        } // i
    } // main
} // main
Summary

Here we learned:

- How to declare a few basic variable types, including integers and floating-points of different sizes.
- How to declare arrays and how to initialize them.
- We used arrays and FOR loops to send a message using:
  - A “shaking” LED display (Explore16, optional)
  - A Morse code way of display (PIC32 SK)
Advanced Material

Let’s learn it anyway!
The ++ and -- operators are smart. If the variable they are applied to is a pointer (a variable type that contains a memory address), they actually increase the address by the exact number of bytes required to represent the quantity pointed to.

For example, a pointer to 16-bit integers will increment its address by two, while a pointer to a 32-bit integer will increment its address by four, and so on…

The increment and decrement operators can also be applied inside a generic expression to operate before or after a variable content is fetched.

See a few examples below (assuming the initial conditions a=0 and b=1).

Post-increment

\[
\text{a} = \text{b}++; \quad // \ a = 1, \ b = 2
\]

In this first case a is assigned the value of b first, b is incremented later.

Pre-increment

\[
\text{a} = ++\text{b}; \quad // \ a = 2, \ b = 2
\]

In this second case, b is incremented first and then its (new) value is passed to a.

You probably have learned this from earlier programming courses.
In the PIC16 and PIC18 8-bit architectures, there is a strong incentive to use byte-sized integers wherever possible.

In the PIC32, 32-bit word-sized integers can be manipulated with the same efficiency.

The only limiting factor preventing us from always using 32-bit integers with the MPLAB compilers is the consideration of the relative “scarcity” of the RAM memory.

Another reason to check the Memory Gauge.
More Notes

- **Notes for PIC experts**
  - It can be confusing to see so many different options to represent a loop (while, do, for)
  - In some situations the algorithm will dictate which one to use
  - In most situations there will be a degree of freedom and more than one type of loop might do.
  - Choose the one that makes the code more readable, and if it really doesn’t matter, as in the main loop here, just choose the one you like and be consistent about it.

- **Notes for C experts**
  - Learn to keep the size of your variables to the strict minimum necessary. You will become a better embedded system designer and ultimately.
Tips and Tricks

- It is one of the “crt0” code responsibilities to initialize variable arrays by copying their contents from a table in FLASH memory to RAM, immediately before the main program execution.
- Another useful service performed by the “crt0” code is to initialize every remaining globally declared variable to 0.
- Both operations might require some time to be completed before the main program execution.
- If you have an urgent task to accomplish before the initializations, try using the following function:

  ```c
  void _on_reset(void)
  {
      // something urgent that needs to be done immediately
      // after a reset or at power-up
      // your code here ...
  }
  ```